Florian Grabner florian.grabner@gmx.at

Benutzerdefinierte Funktionen

- Mathematische / Fachliche Inhalte in Stichworten:

  Programmierung mit C

- Kurzzusammenfassung

  Mathcad bietet die Möglichkeit benutzerdefinierte Funktionen in Form von dll's einzubinden. Diese Funktionen haben die selben Eigenschaften wie Mathcad eigene Funktionen.
  Es wird beschrieben wie der Quellcode für solche benutzerdefinierte dll's aufgebaut ist, welche Mathcad spezifischen Features zur Verfügung stehen, wie man zu einer funktionstüchtigen dll kommt und wie man diese in Mathcad richtig einbindet.
  An Hand eines Beispiels werden die besprochenen Dinge demonstriert.

- Lehrplanbezug (bzw. Gegenstand / Abteilung / Jahrgang):

  Angewandte Mathematik, Informatik, alle Abteilungen

- Mathcad-Version:

  Mathcad 11

- Literaturangaben:

  Mathcad Hilfe - Developer's Reference

  Informationen zum verwendeten Compiler (und der Compiler selbst) im Internet unter: microsoft.msdn.com

- Anmerkungen bzw. Sonstiges:

  Grundsätzliche Programmierkenntnisse (in C) werden vorausgesetzt.
  Einführungen in C finden sich im Internet unter:
  Softwareentwicklung mit C, Schmaranz

Inhalt:

1. Allgemeine Bemerkungen

Mathcad bietet zwei unterschiedliche Möglichkeiten Berechnungsschritte zu automatisieren. Zum einen das interne Programmiertool mit dem schnell und unkompliziert in vielen Fällen sehr gute Problemlösungen erzielt werden können. Benutzerdefinierte Funktionen dieser Art werden in dieser Arbeit nicht behandelt (siehe dazu "Programmieren in Mathcad (Eine Einführung)" von Wilfried Rohm). Zum anderen existiert eine externe Programmierschnittstelle über die benutzerdefinierte dlls in Mathcad eingebunden werden können. In dieser Arbeit finden sich alle notwendigen Informationen um eigene dlls für Mathcad zu schreiben.

Konventionen:

- Codefragmente sind in `Courier New` geschrieben.
- Verzeichnisnamen werden in GROSSBUCHSTABEN geschrieben
  (Der Installationspfad z.B: C:\PROGRAMME\MATHSOFT\MATHCAD\ wird mit INSTALL\ abgekürzt.)
- C unterscheidet wie Mathcad zwischen Groß- und Kleinschreibung!

zurück zur Inhaltsangabe

2. Der Quellcode

Der Aufbau des Quellcode kann primär in zwei Abschnitte unterteilt werden.
- Die Funktion(en) und Fehlerbehandlung
- Die Routine zum Einbinden der dll in Mathcad

2.1 Funktionen und Fehlerbehandlung

Datentypen

Besondere Bedeutung kommt neben den C-typischen Datentypen (integer, float, double, ...) den Mathcad spezifischen Datentypen zu, die den Datentransfer zwischen Mathcad und der Funktion erlauben. Wir unterscheiden aufgrund der Art der Werte zwischen Skalare und Arrays (Vektoren, Matrizen).

Skalare

Mathcad kann mit reellen als auch mit komplexen Zahlen arbeiten. Da die reellen Zahlen eine Teilmenge der komplexen Zahlen sind, reicht ein Typ für beide (bei den reellen Zahlen ist der imaginäre Anteil gleich Null). Dies wird über eine Structure mit zwei Elementen realisiert:

```
COMPLEXSCALAR

->real            // Zugriff auf den Realteil; z.B. Zahl->real
->imag            // Zugriff auf den Imaginärteil; z.B. Zahl->imag
```

Die Genauigkeit der beiden Elemente ist mit double festgelegt.
Für den Austausch von Daten zwischen Mathcad und der Funktion verlangt Mathcad spezielle Formate. Diese Definitionen können auch abgekürzt werden. Für Output und Input lauten diese:

Output (Rückgabewert der Funktion):

```
COMPLEXSCALAR * const namedesrückgabewertes
      oder
LPCOMPLEXSCALAR namedesrückgabewertes
```

Input (Argumente der Funktion):

```
const COMPLEXSCALAR * const namedesargumentes
      oder
LPCCOMPLEXSCALAR namedesargumentes
```

Arrays

```
COMPLEXARRAY

->hReal[m][n]     // Zugriff auf Realteil
->hImag[m][n]     // Zugriff auf Imaginärteil
->cols            // Anzahl der Spalten der Matrix
->rows            // Anzahl der Reihen der Matrix
```

Die Structure für Arrays hat zwei zusätzliche Elemente.
Sowohl der Real- als auch der Imaginärteil werden in zweidimensionale Arrays gespeichert. Der Zugriff erfolgt über zwei eckige Klammernpaare wobei im ersten Klammernpaar die Spalte (m) und im zweiten die Zeile (n) steht. Wie in Mathcad selbst gilt auch hier, dass die erste Zeile bzw. Spalte bei Null beginnt (d.h. m geht von Null bis cols - 1 und n geht von Null bis rows - 1)

Wie bei dem skalaren Datentyp erfolgt auch hier der Austausch über spezielle Formate. Analog zu oben gilt:

Output (Rückgabewert der Funktion):

```
COMPLEXARRAY * const namedesrückgabewertes
      oder
LPCOMPLEXARRAY namedesrückgabewertes
```

Input (Argumente der Funktion):

```
const COMPLEXARRAY * const namedesargumentes
      oder
LPCCOMPLEXARRAY namedesargumentes
```

Bei Arrays gilt es noch zu beachten, dass aufgrund des größeren Speicherbedarfs eigens Speicherplatz reserviert werden muss. Dies muss explizit in der Funktion erfolgen bevor die entsprechende Variable verwendet wird. Mathcad bietet dafür eine eigene Funktion an:

```
MathcadArrayAllocate(Matrix, Zeilen, Spalten, Realteil, Imaginärteil)
```

`Matrix`	Zeiger zur `COMPLEXARRAY` Structure (z.B. `namedesrückgabewertes`, für Argumente muss kein Speicherplatz reserviert werden!)
`Zeilen`	Anzahl der Zeilen (z.B. `namedesrückgabewertes->rows`)
`Spalten`	Anzahl der Spalten (z.B. `namedesrückgabewertes->cols`)
`Realteil`	`TRUE` falls Realteil verwendt werdet wird. Ansonsten `FALSE`
`Imaginärteil`	`TRUE` falls Imaginärteil verwendet werden soll. Ansonsten `FALSE` (für reelle Zahlen)

`MathcadArrayAllocate` gibt `TRUE` falls Speicherplatz fehlerfrei reserviert werden konnte bzw. `FALSE` wenn ein Fehler aufgetreten ist zurück.

Wird ein Array als Argument übergeben, dass nur reelle Zahlen enthält (d.h. es existiert kein 2-dim. Array für den Imaginärteil) so wird dem Element `hImag` der Wert `NULL` zugewiesen.

Funktion

Das Schema jeder Funktionsdefinition sieht wie folgt aus.

```
LRESULT InterneNameDerFunktion(Rückgabewert, Argument 1, ... )
{
      /* Quellcode der Funktion */

      return 0;
}
```

Der Funktionsname darf beliebig lang sein und frei gewählt werden jedoch ohne Sonder- und Leerzeichen. Der Name sollte selbsterklärend sein und kann durchaus länger ausfallen, da dieser Funktionsname nur intern, d.h. nur in der dll gebraucht wird (siehe Abschnitt *FUNCTIONINFO*). Betreffend der Datentypen von Rückgabewert und den Argumenten siehe Abschnitt *Datentypen*.

Die eigentliche Funktion steht dann (wie in C üblich) zwischen zwei geschwungenen Klammer. Die letzte `return`-Anweisung zeigt an, dass die Funktion ohne Fehler beendet wurde.

Als Beispiel wollen wir folgende Fragestellung mit den soeben besprochenen Dingen lösen:
Das erste und dritte Element eines 3x1-Vektors sollen miteinander vertauscht werden. Es sei zunächst davon auszugehen, dass die Elemente nur reell sind und der Vektor die geforderte Dimension hat.

```
LRESULT VertauscheElement1UndElement3 (
      LPCOMPLEXARRAY vektornachvertauschen,
      LPCCOMPLEXARRAY vektorvorvertauschen)
{

MathcadArrayAllocate(vektornachvertauschen, 3, 1, TRUE, FALSE);

vektornachvertauschen->hReal[0][0] = vektorvorvertauschen->hReal[0][2];
vektornachvertauschen->hReal[0][1] = vektorvorvertauschen->hReal[0][1];
vektornachvertauschen->hReal[0][2] = vektorvorvertauschen->hReal[0][0];

return 0;
}
```

Der Algorithmus um die Vertauschung durchzuführen ist erwartungsgemäß nicht sonderlich aufwendig und braucht keine Erklärung. Wir sind aber einfach davon ausgegangen, dass ein reeller 3x1-Vektor übergeben wird. Wir sehen sehr schnell, dass die Funktion einen Fehler verursacht wenn beispielsweise ein 2x1-Vektor übergeben wird. Wir sollten den Quellcode also um sog. Fehlerbehandlungsroutinen erweitern um solchen Eventualitäten vorzubeugen.

Fehlerbehandlung

Um sicherzugehen, dass die selbst programmierte Funktion nur sinnvolle Ergebnisse liefert ist es ratsam Fehlerbehandlungsroutinen zu integrieren. Man sollte schon vor dem eigentlichen schreiben des Quellcodes wissen welche Fehler bzw. Eventualitäten auftreten können und welche man mit einer entsprechenden Routine abfangen möchte. Hier eine kurze Liste (nicht vollständige) von möglichen und sinnvollen Abfragen:

- Benutzer hat Berechnung abgebrochen (sinnvoll bei langen Rechengängen)
- Ist genügend Speicher vorhanden (sollte zur Sicherheit in jedem Programm stehen)
- Sind die Argumente reell / komplex (sollte vor der Berechnung je nach Fall abgefragt werden)
- Hat die Matrix die richtige Dimension (wenn bestimmte Art gefragt ist, z.B. Vektor)

Mathcad selbst fängt die folgenden Fehler automatisch ab: Division durch Null, Overflow und ungültige Operationen. Für diese Fälle braucht keine eigene Fehlerbehandlung vorgesehen werden. Für die anderen Fälle besteht die Fehlerbehandlung im Quellcode aus einem Fehlermitteilungsverzeichnis (error message table) und einer einfache Abfrage an entsprechender Stelle im Code.

Das Fehlermitteilungsverzeichnis muss ganz am Anfang definiert werden und hat folgenden Aufbau (für i Fehlermeldungen):

```
char * NameDesErrorMessageTable[i] ={
      "Fehlermeldung 1",
      "Fehlermeldung 2",
      ....
      "Fehlermeldung i"
};
```

Das Fehlermitteilungsverzeichnis kann auch anders heißen. Es muss dann in der Dll Entry Point Routine der entsprechende Name verwendet werden (siehe Abschnitt *DLL Entry Point Routine*).

Die Abfrage erfolgt an entsprechender Stelle im Programm mit einer einfachen `if`-Bedingung und der `return`-Anweisung. Der Rückgabewert ist entweder eine Zahl, die für die Position der Fehlermeldung im Fehlermitteilungsverzeichnis steht oder der `MAKELRESULT(m,n)`-Anweisung wenn die Fehlermeldung Nummer m unter dem nten Argument erscheinen soll.

Die Funktion aus Abschnitt *Funktion* sieht mit Fehlerbehandlungsroutinen wie folgt aus:

```
char * myErrorMessageTable[3] = {
      "Der Vektor muss reell sein!",
      "Der Vektor muss muss die Dimension 3x1 haben",
      "Nicht genügende Speicherplatz vorhanden"
};

LRESULT VertauscheElement1UndElement3 (
      LPCOMPLEXARRAY vektornachvertauschen,
      LPCCOMPLEXARRAY vektorvorvertauschen)
{
// Überprüfe ob vektorvorvertauschen die richtige Dimension hat

      if ( vektorvorvertauschen->cols != 1 ||
           vektorvorvertauschen->rows != 3 )
            return MAKELRESULT( 2 , 1 );

// Überprüfe ob vektorvorvertauschen reell ist

      if ( vektorvorvertauschen->hImag != NULL )
            return MAKELRESULT( 1 , 1 );

// Überprüfe ob genügend Speicher für vektornachvertauschen vorhanden

      if ( !MathcadArrayAllocate(vektornachvertauschen,
                                 3, 1, TRUE, FALSE) )
            return 3;

// Sind keine Fehler aufgetreten führe Berechnung durch

vektornachvertauschen->hReal[0][0] = vektorvorvertauschen->hReal[0][2];
vektornachvertauschen->hReal[0][1] = vektorvorvertauschen->hReal[0][1];
vektornachvertauschen->hReal[0][2] = vektorvorvertauschen->hReal[0][0];

return 0;
}
```

Bei längeren Operationen (z.B. Schleifen) kann es von Nöten sein einen eventuellen Abbruch von Seiten des Benutzers mit einer Fehlerbehandlungsroutine abzufangen. Zu diesem Zweck bietet Mathcad eine eigene Funktion an:

```
isUserInterrupted()
```

Diese Funktion hat keine Argumente und gibt `TRUE` zurück falls der Benutzer die [ESC]-Taste drückt.

Wird mit Arrays gearbeitet und es kommt zu einem Abbruch der Funktion, entweder durch Abbruch des Benutzers oder durch eine andere Fehlerbehandlungsroutine, so ist es sinnvoll den zuvor reservierten Speicher für das Array wieder an das Betriebssystem zurück zu geben. Dies geschieht mit der Funktion:

```
MathcadFree(Addresse)
```

`Adresse` ist der Zeiger zur Structure `COMPLEXARRAY` (z.B. `namedesrückgabewertes`).

2.2 Routine zum Einbinden der Dll in Mathcad

FUNCTIONINFO

Diese Structure enthält alle Informationen die Mathcad braucht um die Funktion verfügbar zu machen. Folgende Informationen müssen enthalten sein:

```
FUNCTIONINFO funktionsname ={
```

Der Name `funktionsname` entspricht dem Namen, unter dem die Funktion in Mathcad verfügbar sein soll.

```
"funktionsname",              // Name der Funktion in Mathcad
"Argument1, Argument2",       // Beschreibung der Argumente
"Beschreibung der Funktion"   // Beschreibung der Funktionsweise
```

Diese drei Einträge dienen zur Beschreibung der Funktion.

```
(LPCFUNCTION)InterneNameDerFunktion,
```

Dieser Eintrag verweist auf den ausführtbaren Code, der die Funktion definiert (siehe Abschnitt Funktion)

```
COMPLEX_ARRAY,    // Typ des Rückgabewertes (oder COMPLEX_SCALAR)
2,                // Anzahl der Argumente (z.B. 2 - Arg1, Arg2 )
{COMPLEX_SCALAR, COMPLEXARRAY}
};
```

Der letzte Eintrag gibt in einer Liste die Typen der Argumente an. In diesem Beispiel wäre Arg1 ein Scalar und Arg2 ein Array. Die Structure wird wieder mit geschwungender Klammer und Strich-Punkt geschlossen. Für unser Beispiel hat die `FUNCTIONINFO` Structure folgende Form (wenn die Funktion in Mathcad unter dem Namen VTausch verfügbar sein soll) :

```
FUNCTIONINFO VTausch = {
      "VTausch",
      "V",
      "Vertauscht das erste und dritte Element eines 3x1-Vektors V",
      (LPCFUNCTION) VertauscheElement1UndElement3,
      COMPLEX_ARRAY,
      1,
      {COMPLEX_ARRAY}
};
```

DLL Entry Point Routine

Bevor Mathcad die benutzerdefinierte Funktion verwenden kann muss sie registriert werden. Dies geschieht während die dll gelanden wird über die sog. DLL Entry Point Routine. Diese Routine ist vom verwendeten Compiler abhängig. Der folgende Quellcode gilt für den MS Visual C++ Compiler (siehe Kapitel 3):

```
BOOL WINAPI _CRT_INIT(HINSTANCE hinstDLL, DWORD dwReason, LPVOID
lpReserved);

BOOL WINAPI  DllEntryPoint (HINSTANCE hDLL, DWORD dwReason, LPVOID
lpReserved)
{
      switch (dwReason)
      {
            case DLL_PROCESS_ATTACH:
                  if (!_CRT_INIT(hDLL, dwReason, lpReserved))
                        return FALSE;
```

```
                        if ( CreateUserErrorMessageTable(
                              hDLL,ANZAHL_DER_FEHLER,myErrorMessageTable))
                                    CreateUserFunction( hDLL,
                                                        &funktionsname );
                              break;

                  case DLL_THREAD_ATTACH:
                  case DLL_THREAD_DETACH:
                  case DLL_PROCESS_DETACH:
                        if (!_CRT_INIT(hDLL, dwReason, lpReserved))
                              return FALSE;
                        break;
            }
            return TRUE;
      }
```

Für die genaue Erklärung dieser Sequenz verweise ich auf die Hompage von Microsoft für Softwareentwicklung (microsoft.msdn.com). Dieser Codeschnipsel kann direkt in den eigenen Quellcode übernommen werden, wenn man die rot markierten Passagen anpasst. Zu diesem Zweck sehen wir uns die erste grün hervorgehobene Funktion an.

```
CreateUserErrorMessageTable
```

Diese Funktion registriert das Fehlermitteilungsverzeichnis. Das erste Argument ist für die Handhabung der dll und wird von der DLL Entry Point Rountine bereitgestellt. Das zweite Argument steht für die Anzahl der Fehlermitteilungen. Das dritte Argument ist der Name der für das Fehlermitteilungsverzeichnis vergeben wurde (siehe Abschnitt *Fehlerbehandlung*). Diese Zeile würde für unsere Beispiel wie folgt aussehen:

```
CreateUserErrorMessageTable ( hDLL, 3, myErrorMessageTable )
```

Die Funktion gibt `TRUE` zurück wenn die Registrierung erfolgreich war.
Ist das der Fall, dann wird die zweite grün hervorgehobene Funktion aufgerufen.

```
CreateUserFunction
```

Diese Funktiuon registriert die Funktion in Mathcad. Das erste Argument dient wieder der Handhabung der dll. Das zweite Argument hingegen zeigt zu der zuvor definierte `FUNCTIONINFO` Structure. Wichtig ist das &-Zeichen (ermöglicht den Zugriff auf dei Adresse in C). Für unser Beispiel gilt:

```
CreateUserFunction ( hDLL, &VTausch )
```

zurück zur Inhaltsangabe

3. Kompilieren und Einbinden von Dlls

Kompilieren und Linken

Wie schon im Abschnitt *DLL Entry Point Routine* angedeutet, verwende ich für diese Arbeit den Visual C++ Compiler von Microsoft. Theoretisch kann jeder 32-bit Compiler verwendet werden, wenn etwaige Änderungen am Quellcode vorgenommen werden. Es gibt aber zwei gute Gründe warum ich in dieser Arbeit den Microsoft Compiler verwende. Zum einen kann kann der Compiler und die nötigen Bibliotheken gratis vom Internet heruntergeladen werden (siehe dazu *Anhang A*) und zum anderen gibt es einige Quellcodebeispiele von Mathcad (im Verzeichnis INSTALL\USEREFI\MICROSOFT) für diesen Compiler (Mathcad 2001 bietet diese Beispiele auch für andere Compiler an. Die Version 12 beschränkt sich allerdings nur mehr auf Microsoft).

Zum Kompilieren braucht man neben der Quellcodedatei auch noch die Header-Datei "Mcadincl.h" (befindet sich im Verzeichnis INSTALL\USEREFI\MICROSOFT\INCLUDE) sowie die Libary-Datei "Mcaduser.lib" (befindet sich im Verzeichnis INSTALL\USEREFI\MICROSOFT\LIB). Es bietet sich an die Dateien in ihren Verzeichnissen zu belassen und die Quellcodedatei einfach in den Order SOURCES zu kopieren

Microsoft Visual C++ Toolkit 2003

Wie im *Anhang A* erwähnt erfolgt hier die Eingabe über die Kommandozeile. Es empfiehlt sich deshalb die folgenden Befehle in ein BAT-File zu speichern und dieses auszuführen.
Ausgehend von der soeben beschriebenen Situation starten sie das MS Visual C++ Toolkit. Wechseln sie in das Verzeichnis in dem sich die Quellcode Datei befindet (z.B. INSTALL\USEREFI\MICROSOFT\SOURCES). Geben sie die folgenden beiden Befehlzeilen ein.

```
cl -c -I..\INCLUDE -DWIN32 File.c
link -out:File.dll -dll -entry:"DllEntryPoint" File.obj
     ..\LIB\MCADUSER.LIB
```

Der Befehl `cl` ruft den Compiler auf. `File.c` ist der Name der Quellcode-Datei. Die Option bedeuten

`-c`	Der Quellcode wird nur kompiliert und nicht gelinkt (es entsteht eine Datei mit Endung `.obj`)
`-I`	Gibt den Pfad zu dem Verzeichnis an in dem sich die notwenigen Include-Files befinden. Pfad kann relativ oder absolut angegeben werden, z.B: die Datei `Mcaduser.h` befindet sich im Verzeichnis INSTALL\USEREFI\MICROSOFT\INCLUDE\ so kann man schreiben ..\INCLUDE (wenn man sich Im Verzeichnis INSTALL\USEREFi\MICROSOFT\SOURCES befindet - relative Schreibweise) oder INSTALL\USEREFI\MICROSOFT\INCLUDE (absolute Schreibweise)
`-DWIN32`	steht für eine 32bit-Windows Anwendung.

Der Befehl `link` ruft den Linker auf. `File.obj` bzw. `Mcaduser.lib` sind die zu linkenden Dateien (mit Pfad).
Optionen:

`-out:`	Umbenennen der Output-Datei
`-dll`	Es soll eine dll erstellt werden
`-entry`	Name des DLL Entry Point der im Quellcode definiert werden muss (vgl. *Kapitel 2 Dll Entry Point Routine* - hier ist der Name einfach DllEntryPoint).

Achtung: Der Zeilenumbruch der den Befehl `link` i n zwei Zeilen aufteilt, wurde nur aus Platzgründen eingeführt. Bei der Eingabe bedeutet das drücken der Return-Taste, dass der Befehl ausgeführt wird!

Microsoft Visual C++ Express

Aufgrund der grafischen Entwicklungsumgebung ist die Handhabung wesentlich eleganter. Man kann das Programm auch benutzen um den Quellcode zu schreiben. Zum Erstellen der dll geht man wie folgt vor:

1. Öffne das Programm
2. Menü *File / New / Project* ..
3. Wähle: *Project Types - Visual C++ / WIN32* und *Templates: Console Application (WIN32)*
   Gib einen Namen ein und wähle einen Zielordner (Location).
4. Klicke auf *Application Settings* und wähle *Application Typ - DLL , Applicational option - Empty Project*
5. Klicke mit der Rechten Maustaste auf *Source Files* (im Solution Explorer) - *Add / New Item* .. (oder *Existing Item* ... falls die Quellcode-Datei bereits existiert, dann entfallen die Schritte 6 und 7)
6. Wähle: *Catagories - Visual C++* und *Templates - CPP File (.cpp)*. Gibt Name mit Endung .c ein (.c für C Source File)
7. Schreiben und abspeichern des Quellcodes.
8. Klicke mit Rechter Maustaste auf den Project Name (im Solution Explorer) und wähle *Properties*.
9. Wähle *C/C++ - General* und füge das Verzeichnis unter *Additional Include Directiories* ein in dem sich die Datei Mcaduser.h befindet.
10. Wähle *Linker - General* und füge das Verzeichnis unter *Additional Library Directiories* ein in dem sich die Datei Mcaduser.lib befindet.
11. Wähle *Linker - Input* und füge die Datei Mcaduser.lib unter *Additional Dependencies* ein.
12. Wähle *Linker - Advanced* und füge den Name des DLL Entry Points unter *Entry Point* ein (z.B. DllEntryPoint)
13. Schließe das Fenster mit OK.
14. Erstellen der Dll über Menü *Build / Build Solution*.

Einbinden

Um die dll für Mathcad verfügbar zu machen, braucht man sie nur in den Ordner INSTALL\USEREFI kopieren. Alle dlls in diesem Verzeichnis werden bei jedem Programmstart automatisch geladen. Sollten sie Mathcad gestartet haben wenn sie die dll ins USEREFI-Verzeichnis kopieren, so muss Mathcad danach neu gestartet werden.

***Funktion-Einfügen* Fenster**

Wie im Kapitel 2 besprochen kann man die selbst geschriebene Funktion auch über das *Funktion-Einfügen* Fenster aufrufen. Damit die Funktion dort erscheint muss ein Eintrag in die Datei User.xml im Verzeichnis INSTALL\DOC\FUNCDOC vorgenommen werden. Am Anfang dieser Datei findet sich quasi ein Leerformular, dass man nur kopieren und ausfüllen braucht (Das Tag <local_name> kann herausgelöscht werden).

Wenn man für seine Funktion auch eine Hilfedatei verwenden möchte, so kann man diese am Anfang der Datei im Tag <help_file> angeben. Der Verweis auf die jeweilige Hilfeseite erfolgt dann im jeweiligen Eintrag durch das Tag <help_topic>. Die Angabe einer Hilfedatei ist nicht zwingend erforderlich.

Mehrere Funktionen

Es ist auch möglich mehrere Funktionen in einer dll zu definieren. Man muss aber beachten, dass pro dll nur ein Fehlermitteilungsverzeichnis registriert werden kann. Es ist auch möglich jede Funktion in einer eigenen Datei abzuspeichern (bei mehreren Funktionen eine sehr übersichtliche Lösung). Man hat dann mindestens eine Datei in der die Funktion und ihre FUNCTIONINFO-Structure definiert ist und eine weitere Datei mit dem Fehlermitteilungsverzeichnis und der Routine zum registrieren der Funktionen in Mathcad. In dieser Datei müssen die FUNCTIONINFO-Structures über die `extern`-Anweiung geladen werden.
Das Beispiel aus Kapitel 4 ist sowohl auf die konventionelle Methode (alles in einer Datei) alsauch in der Multidatei Variante implementiert.

zurück zur Inhaltsangabe

4. Ein Beispiel

Ein Beispiel wurde bereits im Kapitel 3 diskutiert. Der komplette Quellcode für dieses Beispiel ist dieser Arbeit beigefügt. In diesem Kapitel möchte ich aber die Aufgabenstellung eines etwas aufwendigeren Beispieles besprechen:

In der Technik ist die Vektorrechnung ein sehr beliebtes Werkzeug um Vorgänge zu beschreiben. So kommt es häufig vor, dass ein Vektor in einem Koordinatensystem gedreht werden muss bzw. ein Koordinatensystem gedreht werden soll (z.B: zur Ermittlung der Spannungshauptachsen in der Mechanik). Dies geschieht mit Hilfe einer sog. Drehmatrix indem man den zu drehenden Vektor mit der Drehmatrix multipliziert. Die allgemeine Form der Drehmatrix kann aus mathematischen Handbüchern (z.B. Bronstein - Taschenbuch der Mathematik) entnommen werden.

Die Aufgabenstellung besteht darin, eine Drehung um je eine Koordinatenachse eines kartesischen Koordinatensystems mit einer selbst geschriebenen Funktion zu automatisieren. Es sollen vier Funktionen definiert werden, die folgende Funktionalität aufweisen sollen:

Allgemeine Hinweise:

Der zu drehende Vektor soll die Dimension 3x1 haben.
Es sollen nur reelle Vektoren bzw. Winkel akzeptiert werden.
Die trigonometrischen Funktionen sind in der Header Datei `math.h` definiert und müssen mit einer `include`-Anweisung eingebunden werden. Die Schreibweise der Funktionen ist: `sin( ), cos( )`

Drehung um die X-Achse:

Funktionsname: `DrehX(V,W)`
Argumente: `V` - der zu drehende Vektor
`W` - Winkel in dem im positiven Sinn gedreht werden soll.

Die entsprechende Drehmatrix hat folgende Form:

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(W) & -\sin(W) \\ 0 & \sin(W) & \cos(W) \end{pmatrix}$$

Drehung um die Y-Achse:

Funktionsname: `DrehY(V,W)`
Argumente: `V` - der zu drehende Vektor
`W` - Winkel in dem im positiven Sinn gedreht werden soll.

Die entsprechende Drehmatrix hat folgende Form:

$$\begin{pmatrix} \cos(W) & 0 & \sin(W) \\ 0 & 1 & 0 \\ -\sin(W) & 0 & \cos(W) \end{pmatrix}$$

Drehung um die Z-Achse:

Funktionsname: `DrehZ(V,W)`
Argumente: `V` - der zu drehende Vektor
`W` - Winkel in dem im positiven Sinn gedreht werden soll.

Die entsprechende Drehmatrix hat folgende Form:

$$\begin{pmatrix} \cos(W) & -\sin(W) & 0 \\ \sin(W) & \cos(W) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Drehung um alle drei Achsen:

Funktionsname: `Dreh(V,XW, YW, ZW)`
Argumente: `V` - der zu drehende Vektor
`XW` - Winkel in dem im positiven Sinn um die X-Achse gedreht werden soll.
`YW` - Winkel in dem im positiven Sinn um die Y-Achse gedreht werden soll.
`YW` - Winkel in dem im positiven Sinn um die Z-Achse gedreht werden soll.

Die Reihenfolge der Drehung sei wie folgt: X,Y,Z

Anmerkung: Verwende die oben definierten Funktionen.

Die vollständige Implementierung sowie die kompilierte dll ist dieser Arbeit beigefügt.

zurück zur Inhaltsangabe

Anhang A

Visual C++ Toolkit 2003

Link: **http://msdn.microsoft.com/visualc/vctoolkit2003/** (ca. 31 MB)

Dieses Programmpaket includiert den MS Visual C++ Compiler wobei die Bedienung nur über die Kommandozeile möglich ist (d.h. keine grafische Entwicklungsumgebung) und es enthält keine Dokumentation.

Um das Setup-File herunter zu laden braucht man nur den obigen Link zu verwenden. Ich empfehle die manuelle Installation, d.h. das Setup-File zuerst auf der Festplatte speichern und erst danach selbst ausführen. Um zur tatsächlichen Download Seite zu kommen braucht man ein .net-Passwort (ist man noch nicht im Besitz eines solchen, gibt es die Möglichkeit sich kostenlos zu registrienen - nur e-mail Adress notwendig)
Zum Installieren einfach das Setup-File ausführen und den Installationsanweisungen folgen.

Um eine dll für Mathcad zu erzeugen muss man noch die Platform SDK (siehe weiter unten) einbinden. Dazu wechselt man ins Installationsverzeichnis, z.B:

```
C:\Programme\Microsoft Visual C++ Toolkit 2003
```

Dann öffnet man die Datei "vcvars32.bat" mit Rechter Maustaste - Bearbeiten
Diese Datei wird bei jeder Sitzung automatisch geladen und setzt die Umgebungsvariablen PATH, INCLUDE und LIB. Wir fügen in der vierten Zeile (Set INCLUDE=.....) den Pfad der Include-Files der Platform SDK hinzu, z.B:

```
Set INCLUDE=D:\Microsoft Visual C++ Toolkit 2003\include;
      C:\Programme\Microsoft Platform SDK\Include;%INCLUDE%
```

Nach abspeichern der Änderung ist das Visual C++ Toolkit richtig konfiguriert.

Visual C++ Express Beta

Link: **http://lab.msdn.microsoft.com/express/visualc/** (ca. 67 MB)

Dieses Programm ist Teil von MS Visual Studio. Es bietet eine grafische Entwicklungsumgebung und es gibt die Möglichkeit eine Dokumentation herunter zu laden.

Um Visual C++ Express Beta zu verwenden muss zuerst Microsoft .NET Framework 2.0 Beta installieren.
Man wählt wieder die manuelle Installation und kommt dann auf die Seite mit den Download Anweisungen. Hier finden sich die entsprechenden Download Link für Microsoft .NET Framework 2.0 Beta, Visual C++ 2005 Express Edition Beta und Microsoft MSDN Express Library 2005 Beta (enthält die Dokumentaion). Um Visual C++ Express Beta herunter zu laden braucht man wieder ein .net-Passwort (vgl. Visual C++ Toolkit 2003).

Zum Installieren führt man zuerst das Setup-File für das Microsoft .NET Framework 2.0 Beta aus und folgt den Installationsanweisungen. Danach installiert man Visual C++ 2005 Express Edition Beta durch das selbe Vorgehen. Bei Bedarf verfahrt man mit der Microsoft MSDN Express Library 2005 Beta gleich.

Um Visual C++ 2005 Express Edition Beta mit der Platform SDK zu verknüpfen geht man wie folgt vor:

1. Starten von Visual C++ 2005 Express Edition Beta über das Startmenü
2. Öffnen des Menüs: *Tools / Options ...*
3. Erweitern der Kategorie *Projects and Solution* und makieren des Punktes *VC++ directories*
4. Wähle im Pull-down Menü *Show directories for:* den Punkt *Include files*
5. Klicke auf den Button *Insert line*
6. Gib den Pfad zum Include-Ordner der Platform SDK an (durch klicken des Button mit den drei Punkten kann man den Ordner auch wählen), z.B: D:\MICROSOFT PLATFORM SDK\INCLUDE
7. Schließe das Fenster durch klicken des Buttons *OK*

Platform SDK

Link: **http://www.microsoft.com/msdownload/platformsdk/sdkupdate/downlevel.htm** (insgesamt ca. 233 MB)

SDK steht für Software Development Kit und enthält eine Vielzahl von Include-Files, Libaries und Quellcode Beispiele die für das erstellen von Programmen unter Windows eigentlich unentbährlich sind (z.B. Windows.h).

Die Platform SDK kann unter dem obigen Link herunter geladen werden. Es müssen alle 10 cab-Files, das bat-File sowie das sog. Extraction Utility File heruntergeladen werden.

Um Platform SDK installieren zu können muss man die cab-Files extrahieren. Dazu geht man wie folgt vor:
1. Öffne die Eingabeaufforderung (Startmenü / Programme / Zubehör)
2. Wechsle in das Verzeichns in dem sich die soeben heruntergeladenen Dateien befinden (Verwende den DOS-Befehl `cd` , z.B. `cd C:\TEMP` um ins Verzeichnis TEMP auf C zu gelangen).
3. Gib nun folgenden Befehl ein: `psdk-full Verzeichnisname` wobei `Verzeichnisname` für ein beliebiges Verzeichnis steht in das extrahiert werden soll, z.B. C:\TEMP

Zum installieren wechselt man in das Verzeichnis in das soeben extrahiert worden ist und führt das Setup-File aus. Folge den Installationsanweisungen. Man braucht nur den Microsoft Windows Core SDK zu installieren (abwählen aller anderen Optionen)

zurück zur Inhaltsangabe